Address Translation Services Revision 0.7

March 27, 2006

REVISION	REVISION HISTORY	DATE
0.3	Initial release draft specification	9/2005
0.5	Initial release of the final requirements	10/12/2005
0.7	Initial release of the specification methods	1/30/2006
0.9		

The PCI-SIG disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does the PCI-SIG make a commitment to update the information contained herein.

Contact the PCI-SIG office to obtain the latest revision of the specification.

Questions regarding this document or membership in the PCI-SIG may be forwarded to:

Membership Services
http://www.pcisig.com
E-mail: administration@pcisig.com
Phone: 503-619-0569
Fax: 503-644-6708

Technical Support
www.pcisig.com/developers/technical_support

DISCLAIMER

Contents

Figures

Tables

Preface

This specification describes the extensions required to allow PCI Express Endpoints to interact with an address translation agent (TA) in or above a Root Complex (RC) to enable translations of DMA addresses to be cached in the Endpoint. The purpose of having an Address Translation Cache (ATC) in an Endpoint is to minimize latency and to provide a scalable distributed caching solution that will improve I/O performance while alleviating TA resource pressure

Document Organization

The specification is organized into four sections:

1) Introduction and Architectural Overview of the Address Translation Service (ATS) – This section covers the problem space and associated approach to solving this space including ATS operations.
2) ATS TLP Messages and Associated Semantics – This section provides a detailed discussion of the ATS TLP messages their operational semantics.
3) ATS Invalidation Protocol – This section provides a detailed discussion of the ATS invalidation protocol with a number of implementation notes to enable developers implementing to this specification.
4) ATS Configuration – This section provides a detailed discussion of ATS configuration, how to enable an ATC, etc..

Documentation Conventions

Capitalization

Some terms are capitalized to distinguish their definition in the context of this document from their common English meaning. Words not capitalized have their common English meaning. When terms such as "memory write" or "memory read" appear completely in lower case, they include all transactions of that type.
Register names and the names of fields and bits in registers and headers are presented with the first letter capitalized and the remainder in lower case.

Numbers and Number Bases

Hexadecimal numbers are written with a lower case "h" suffix, e.g., 0FFFh and 80h. Hexadecimal numbers larger than four digits have a space dividing each group of four digits, as in 1E FFFF FFFFh.

Binary numbers are written with a lower case "b" suffix, e.g., 1001b and 10b. Binary numbers larger than four digits are written with a space dividing each group of four digits, as in 1000 0101 0010b. All other numbers are decimal.

Reference Information

Reference information is provided in various places to assist the reader and does not represent a requirement of this document. Such references are indicated by the abbreviation "(ref)". For example, in some places, a clock that is specified to have a minimum period of 400 ps also includes the reference information maximum clock frequency of "2.5 GHz (ref)".

Requirements of other specifications also appear in various places throughout this document and are marked as reference information. Every effort has been made to insure that this information accurately reflects the referenced document; however, in case of a discrepancy, this document takes precedence.

Implementation Notes

Implementation Notes should not be considered to be part of this specification. They are included for clarification and illustration only. Implementation Notes within this document are enclosed in a box and set apart from other text.

Discussion Notes

Items enclosed within square brackets ('[ ]') and colored green are strictly informative and serve to provide some insight into the discussions that have taken place in the working group. All items in green are under investigation.

Terms and Abbreviations

Translation Agent (TA)	A logical entity that converts addresses expressed in terms of one address space into an address in a different address space. A Translation Agent (TA) is associated with memory that contains translation tables that the TA will reference for address translation (See ATPT). A TA may contain an Address Translation Cache. A TA may be implemented either in hardware, software, or a combination of both. A TA is treated as implementation-specific and, therefore, is outside the scope of this specification.
Address Translation Cache (ATC)	A hardware entity that stores recently used address translations. This term is used instead of Translation Look-aside Buffer (TLB) to differentiate the TLB used for I/O from the TLB used by the CPU. Each TA is expected to have an ATC co-located with it, but an ATC need not be co-located with a TA.
Address Translation and Protection Table (ATPT)	The data structure(s) accessed by a Translation Agent to determine the mapping of untranslated DMA addresses into translated addresses. The data structure may contain fields that indicate the protection attributes of the translation entry.
Address Translation Services (ATS)	The set of configuration, wire protocol, ATC, etc. required to deliver an address translation solution.
Smallest Translation Unit (STU)	The increment for translation and invalidation. This value is expressed in terms of 4096 bytes units and is programmed into a configuration register on the Function.
Untranslated Address	An address that is formed using the existing programming mechanisms of PCI Express
Translated Address	An address formed by using the results of an address Translation Request.

.

1

1. Architectural Overview

Most contemporary system architectures make provisions for translating addresses from DMA (bus mastering) I/O Endpoints. In many implementations, it has been common practice to assume that the physical address space seen by the CPU and by an I/O Endpoint is equivalent. While in others, this is not the case. The address programmed into an I/O Endpoint is a 'handle' that is processed by the Root Complex (RC). The result of this processing is often a translation to a physical memory address within the central complex. Typically, the processing includes access rights checking to insure that the DMA Endpoint is allowed to access the referenced memory location(s).

The purposes for having DMA address translation vary and include:

- Limiting the destructiveness of a 'broken' or miss-programmed DMA I/O Endpoint;
- Providing for scatter/gather;
- Address space conversion (32-bit I/O Endpoint to larger system address space); and
- Virtualization support.

Irrespective of the motivation, the presence of DMA address translation in the host system has certain performance implications for DMA accesses.

Depending on the implementation, DMA access time can be significantly lengthened due to the time required to resolve the actual physical address. If an implementation requires access to a main-memory-resident translation table, the access time can be significantly longer than the time for an un-translated access. Additionally, if each transaction requires multiple memory accesses (e.g., for a table walk), then the memory transaction rate (i.e. overhead) associated with DMA can be high.

To mitigate these impacts, designs often include address translation caches in the entity that performs the address translation. In a CPU, the address translation cache is most commonly referred to as a translation look-aside buffer (TLB). For an I/O TA (Translation Agent), the term address translation cache or ATC is used to differentiate it from the translation cache used by the CPU.

While there are some similarities between TLB and ATC, there are important differences. A TLB serves the needs of a CPU that is nominally running one thread at a time. The ATC, however, is generally processing requests from multiple I/O Endpoints, each of which can be considered a separate thread. This difference makes sizing an ATC difficult depending upon cost models and expected technology re-use across a wide range of system configurations.

The mechanisms described in this specification allow an I/O Endpoint to participate in the translation process and provide an ATC for its own memory accesses. The benefits of having an ATC within an Endpoint include:

1) Ability to alleviate TA resource pressure by distributing address translation caching responsibility (reduced probability of 'thrashing' within the TA).

2) Enable ATC Endpoints to have less performance dependency on a system's ATC size.
3) Potential to insure optimal access latency by sending pre-translated requests to central complex.

This specification will provide the interoperability that allows PCI Express Endpoints to be use in conjunction with TA – a TA is treated as implementation-specific and is outside the scope of this specification. While it may be possible to implement ATS within other PCI Express Components, this specification is confined to PCI Express Endpoints and PCI Express Root Complex Integrated Endpoints.

1.1. Address Translation Services (ATS) Overview

ATS consists of a set of transactions to enable a downstream Endpoint to:

1) Request a translation of a given address.
2) To indicate that a DMA Read or DMA Write has been translated thereby enabling the ATPT to be bypassed if desired.
3) Indicate that a Translation Invalidation Request has been completed.

ATS consists of a set of transactions to enable a RC to:

1) Send a translation in response to a Translation Request from an Endpoint.
2) Issue a Translation Invalidate Request for a given address or range of addresses.
3) Ascertain whether a Translation Invalidation has been processed by an Endpoint.

ATS base requirements:

1) ATS TLPs must be able to be routed via existing PCIe Base specification methods. Address-based, and Requester ID based routing must be supported.
2) Must allow use of the existing PCIe 1.1 Base specification ordering rules.
3) ATS TLPs need not be treated differently than existing PCIe 1.1 Base specification TLPs in terms of TC management.
4) ATS must not change TC configuration, TC-to-VC mapping, or any of the arbitration schemes defined in the PCIe 1.1 Base specification and associated addendums.
5) Must operate over existing PCIe 1.1 Base specification compliant Switches
6) ATS must support a variety of page sizes.
7) Page size must be a power of 2, naturally aligned.
8) Minimum supported page size is 4096 Bytes. ATC-based components must support this minimum page size.
9) An Endpoint will be informed of the minimum translation or invalidate size it will be required to support to enable the Endpoint an opportunity to optimize its resource utilization. The smallest minimum translation size will be 4096 Bytes.

10) Translate Completion transactions must indicate the range the translation covers.

11) An ATC presents an interface to the system and an ATC is associated with a single function. If the implementation of an ATC on a multi-function Endpoint shares resources among the ATCs, then the logical behavior visible at the PCIe interface shall be consistent with fully independent ATC implementations or SI.

12) ATS transactions must not rely upon the address field of a memory request to communicate additional information beyond its current use as defined by the PCI-SIG.

Implementation Note:

It is likely that the untranslated and translated address range will overlap, perhaps in their entirety. This is not a requirement of ATS but may be an implementation constraint on the TA so that memory requests will be properly routed.

13) From a software perspective, an ATS Root Complex Integrated Endpoint must behave the same as a non-integrated ATS Endpoint.

14) A downstream component may intermix translated and untranslated requests. A method must exist to indicate whether a given TLP request has been translated.

1.2. ATS Operations

Basic operational flow example of an ATS capable Endpoint:

1) A downstream Endpoint is requested to process a given work request. For example, a device driver would program the Endpoint to issue a DMA Read of a specified memory region.

2) The Endpoint would determine whether a translation already exists in its local cache of translations.

    a. If it exists, the Endpoint would issue a DMA Read request with an indication that the referenced address has been translated.

    b. If it does not exist, the Endpoint would issue an ATS Translation Request. Upon receiving an ATS Translation Completion, the Endpoint would issue a DMA Read request with an indication that the referenced address has been translated

3) The RC would receive the DMA Read request and note the address has been translated and therefore does not need to access the ATPT. The RC would service the request and return an appropriate completion.

[Insert flow diagrams to illustrate ATS operation for each TLP type]

1.2.1. Translation Request

A Translation Request comes from an Endpoint and flows upstream to an RC. The RC is responsible for processing the request and generating an appropriate completion.

A Translation Request must:

1) Include a 64-bit address to be translated;
2) Support a method to uniquely tag a given Translation Request and its completion allowing them to be correctly associated;
3) Be allowed to use any TC per the PCIe 1.1 Base specification.

RC responsibilities

1) Must issue at least one Translation Completion TLP for each Translation Request.
    a) The requested address may not be valid. The RC must issue a Completion with a Completion Status to inform the Endpoint of the error.
2) The Translation Completion shall be in the same TC as the Translation Request.

Downstream component responsibilities

1) Must be able to source at least one Translation Request. May be able source multiple Translation Requests that can be pipelined.
2) Must be able to accept the Translation Completion when it is delivered (no backpressure on the fabric as a result of processing a completion).

Areas of investigation:

4) Diagnostics requirements if any

1.2.2. Translation Completion

An ATS Translation Completion is returned by the TA in response to a Translation Request. The RC is responsible for sourcing Translation Completions. The requesting downstream component is responsible for sinking the Translation Completions.

An ATS Translation Completion must:

1) Include a 64-bit translation value or an indication of a translation failure.
2) Include the range for which the translation applies (number of pages).
3) Support a method to uniquely identify a given completion so that an Endpoint may associate it with a Translation Request. This also allows a Endpoint to have more than on outstanding Translation Request at a time.
4) Support a method for indicating address translation failure
5) Provide for a Translation Completion to be sent in the same TC as the request.

RC responsibilities

1) A RC must issue at least one Translation Completion for each address Translation Request.

2) The requested address may not be valid. The RC must issue a Completion to inform the Endpoint of the error. This error is considered non-fatal, correctable.

   [A translation may fail or a translation may be invalid. If the ATPT used a tree-structure for translation tables a translation would 'fail' if one of the directly-level entries was not present. However, if a value can be found at the page table level, that value might indicate that the translation is invalid for reads or writes to system memory but the contents of the table could be returned and no error indicated.]

3) A RC may pipeline multiple Translation Completions, i.e. a RC may return multiple Translation Completions and that these Translation Completions may be in any order relative to Translation Requests.

Endpoint responsibilities

1) Must be capable of sinking all Translation Completions, i.e. without causing backpressure on the PCIe link.
2) Be able to discard translations that might be 'stale'

1.2.3. Invalidate Request

An ATS Invalidate Request is used to invalidate a previously completed translation. Invalidate Requests are sourced by a RC. A downstream component that contains an ATC translation is responsible for sinking the invalidate request.

An Invalidate Request must:

1) Support invalidation on a per address basis.
2) Include a 64-bit address to be invalidated
3) Include the size of the address range being invalidated (variety of methods to indicate size)
4) Support a method to determine whether a specific invalidation has occurred.
5) Support a method to uniquely identify an Invalidate Request. Example options to be investigated:
   a) Inclusion of a unique identifier within the Invalidate Request which will be reflected in the Invalidation Completion.
   b) Use of the existing PCIe tag identifier.

RC responsibilities

1) An RC is required to uniquely identify an Invalidation Request.

Downstream component responsibilities

1) A downstream component must indicate when a requested invalidation is completed. Variety of methods to be investigated.
2) A downstream component must not indicate an invalidation has completed until all outstanding Read Requests that reference the associated translated address have been retired.

3) A downstream component must insure that the invalidation completion indication to the RC will arrive at the RC after any previously posted writes that use the 'stale' address.

4) A downstream component is NOT required to immediately flush all pending requests upon receipt of an Invalidate Request. If transactions are in a queue waiting to be sent, it is not necessary for the Endpoint to expunge requests from the queue even if those transaction use an address that is being invalidated.

Areas of investigation:

1) Translation protocol could support an "invalidate all" for a given Requester ID to simplify design. This is still open to debate but is worthy of some discussion. I'd like to separate this from a broadcast method which may have some issues, however, for a single Requester ID, I think there is merit in allowing software a simple method to "flush" all potential translations without having to flood the fabric with potentially large number of invalidates.

2) Error semantics

3) Diagnostics requirements, if any

4) Implicit flushing (e.g. on reset or change of BDF).

1.3. ATS Management

The Endpoint must manage its ATC in a constrained manner. Not only must it process the Translation Requests and Translation Completions outlined in the previous sections, it must also operate under other constraints as described here

1.3.1. Cache modification

1) An ATC must only be populated using the ATS protocol, i.e. each entry within the ATC must be filled via an ATS Completion in response to the Endpoint issuing an ATS Request for a given address.

2) An ATC cannot be modified except through the ATS protocol. That is:

   a) Host system software cannot modify the ATC other than through the protocols defined in this specification except to invalidate one or more translations in an ATC (an Endpoint reset would be an example of the operations preformed by software to change the contents of the ATC but a reset is only allowed to invalidate entries).

   b) Host system software cannot use software executing on the Endpoint to modify the ATC.

1.3.2. Address Invalidation

1) When an Invalidate Request is received, an ATC must invalidate all translations that use the un-translated address over the entire range specified.

2) The invalidation mechanism must be designed such that it does not create deadlock exposure. For the current proposal where both invalidation requests and responses travel in

the posted channel, this implies that the protocol must be designed such that a post-to-post Endpoint dependency does not exist.

3) An Endpoint implementing an ATC must have a mechanism to publish sufficient information to characterize its invalidation capabilities.

2

2. ATS Translation Services

A TA does translations. An ATC can cache those translations. If an ATC is separated from the TA by PCI Express, the memory request from an ATC will need to be able to indicate if the address in the transaction is translated or not. The modifications to the memory transactions are described in this section as are the transactions that are used to communicate translations between a remote ATC and a central TA.

2.1. Memory Requests with Address Type

An Endpoint with an ATC can send memory read/write Requests that contain either translated or un-translated addresses. As shown in **Error! Reference source not found.** and **Error! Reference source not found.** below, the Address Type (AT) field is used to indicate the type of address that is present in the request header.

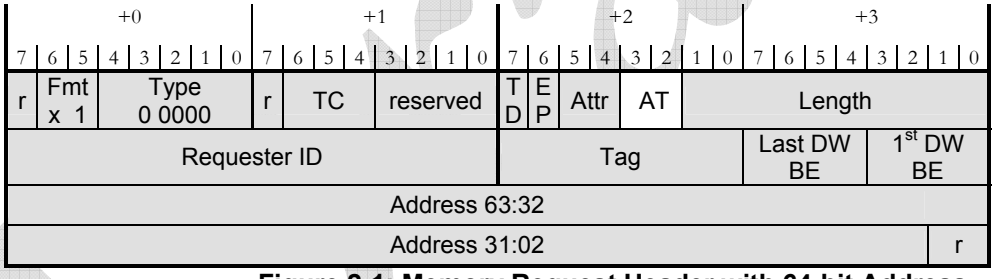


Figure 2-1: Memory Request Header with 64-bit Address

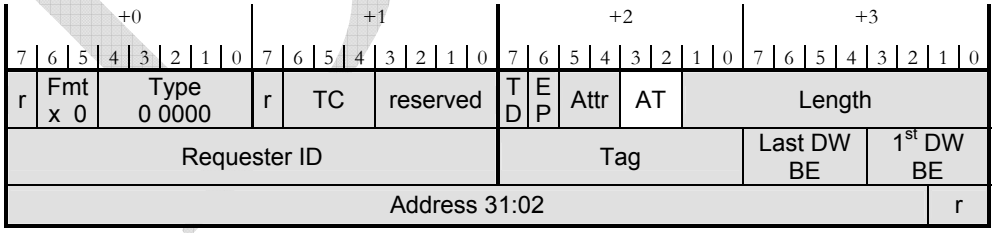


Figure 2-2: Memory Request Header with 32-bit Address

The AT field in the requests is a redefinition of a reserved field in the PCI Express specification. Endpoints that do not implement an ATC will continue to set the AT field to its defined reserved value (0). Endpoints that implement an ATC will set the AT field as follows:

Table 2-1: Address Type (AT) Field Encodings

AT Coding	Mnemonic	Meaning
00b	Default	A TA may treat the address as either virtual or physical.
01b	TranslationRequest	The TA will return the translation of the address contained in the address field of the request as a read completion. This value only has meaning for an explicit Translation Request (see 2.2) and will result in a UR if not in a Translation Request.
10b	Translated	The address in the transaction has been translated by an ATC. If the Function associated with the SourceID is allowed to present physical addresses to the system memory then the TA may not translate this address. If the Function is not allowed to present physical addresses, then the TA may treat this as a UR.
11b	reserved	Results in a UR if used in a Read, Write, or Translation Request.

2.2. Translation Requests

A Translation Request has a format that is similar to that of a memory read. The AT field is used to differentiate a Translation Request from a normal memory read.

The request header for a Translation Request has the following format:

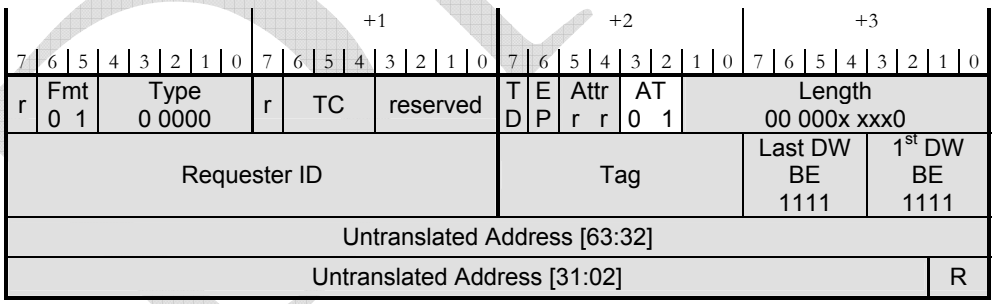


Figure 2-3: Translation Request Header

Translation Requests have the same completion timeout intervals as Read Requests.

2.2.1. Attribute Field

For a Translation Request, the Attr field is reserved for future use and the request will be treated as a Malformed Packet if it is not set to 0. There are no ordering requirements for a Translation Request. A TA may reorder a Translation Request with respect to any other request.

> Implementation Note: Because no ordering can be assumed between Translation Requests and other types of Requests, a Translation Request does not make an effective flushing/ordering primitive.

2.2.2. Length Field

The Length field is set to indicate how many translations may be returned in response to this request. Each translation is 8 bytes in length and represents one or more STUs. The maximum setting for the Length field is the lesser of the sizes determined by the setting of RCB and Max_Read_Request_Size. The Length field in a Translation Request must always indicate an even number of Dwords. If Length is set to indicate a value greater than allowed or if the least-significant bit of the Length field is non-zero, then the TA will treat the request as a Malformed Packet.

If the Length field has a value greater than two, then the Function is requesting translations for a range of memory greater than a single STU. The additional translations, if provided, will be for sequentially increasing, equal sized, STU aligned regions, staring at the requested address.

2.2.3. Tag Field

The Tag field has the same meaning as in a Read Request. The Tag values used in Translation Requests come from the same pool of numbers used for Read Requests.

2.2.4. Untranslated Address Field

A Translation Request includes a 64-bit Address field. This address indicates the address to be translated. The TA will make decisions about the validity of the request, based on the address in the translation request, not the implied subsequent addresses. The TA may return fewer translations than requested but it will not return more.

When multiple translations are requested, the TA will not return a translation if the range of that translation does not overlap the implied range of the Translation Request (this would only apply to translations after the initial value). The implied range of the Translation Request is (STU * (Length/2)).

The Untranslated Address Field in the Translation Request is any address in the range of the first STU. The TA will ignore the low-order bits that are not required to determine the translation.

The Address field in the Translation Request is shown with 52 significant bits. The number of significant bits is actually determined by the setting of the STU configuration parameter. If STU is set to 0 indicating 4KB increments, then only bits 63:12 are significant. If the STU is set to indicate a larger size, then the number of significant bits of the address is adjusted accordingly. The bits in the Address field that are not significant are ignored by the TA and it is not a requirement that they be set to 0.

2.3. Translation Completion

A Translation Completion (either a Cpd or a CplD) is sent by a TA for each Translation Request. This specification describes the meaning of fields in Translation Completions. Fields not defined in this specification have the same meanings proscribed for Read Completions in the PCI Express Base Specification.

If the TA was not able to perform the requested translation, a completion with the following format is used:

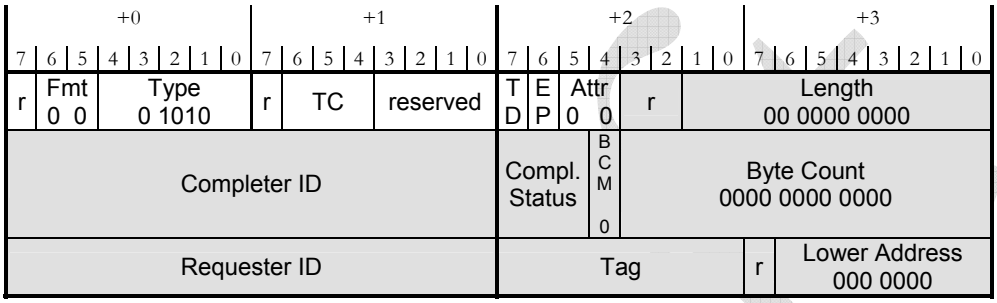


Figure 2-4: Translation Completion with No Data

The values and meaning for the Completion Status field are:

Table 2-2: Translation Completion with No Data Status Codes

Value	Status	Meaning
000b	Success	This completion status has a nominal meaning of "success". The TA will return this value in a Cpl if no table entry is found for the Untranslated Address in the Translation Request. If a table entry is found with R=W=0b, then a CplD is returned instead of a Cpl.
001b	Unsupported Request (UR)	Translation Requests from this Function are not supported by the TA. If an ATC receives this Completion code, it must disable its ATC and not send requests using translated addresses until the ATC is re-enabled. This may also cause an Error Message to be sent if Advanced Error Reporting is supported.
010b	CRS	This value is not allowed in any Completion to a request initiated by a PCI Express Endpoint. If received by an Endpoint it shall be treated as a malformed packet.
100b	Completer Abort (CA)	The TA was not able to translate the address because of an error in the TA. This nominally causes an error to be reported to the device driver associated with the ATC.
All others	Reserved	If a Translation Completion contains a reserved value in the Completion Status field, the Endpoint will treat it as a malformed packet.

Note: Return values other than *Success* indicate an error.

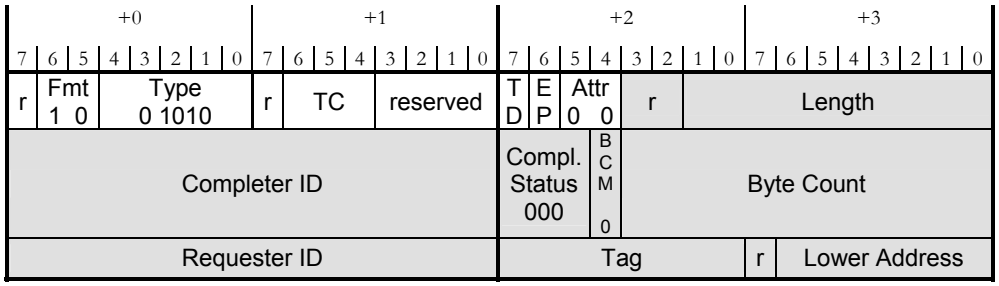


Figure 2-5: Successful Translation Completion

Fields are set in accordance with sections 2.2.9 and 2.3.1 of the PCI Express specification.

Translation completions will be sent using the same TC as the Translation Request.

The Lower Address field will contain a value that will make the packet consistent with RCB semantics. If the results is returned in a single packet, Lower Address will be set to RCB minus Byte Count. If the results is returned in multiple packets, the first packet will have a Lower Address field of RCB minus (Length * 4) and subsequent packets will have a Lower Address field of 00 0000b.

If the translation is successful, the Completion Status field will be 000b and a data payload will follow the header. The contents of the data payload are:

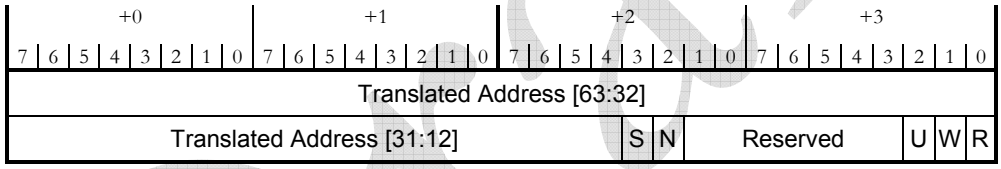


Figure 2-6: Translation Completion Data Entry

Table 2-3: Translation Completion Data Fields

Field	Meaning
S	**Size of translation** – This field is 0b if the translation applies to a 4KB range of memory. If this field is 1b, then the translation applies to a range of memory that is larger than 4KB. See 2.3.1
N	**Non-snooped accesses** – If this field is 1b, then the read and write requests that use this translation must set the No Snoop bit in the Attribute field. If it is zero, then the Endpoint may use other means to determine if No Snoop should be set.
Reserved	These bits shall be ignored by the ATC.
U	**Untranslated access Only** – When this field is set to 1b a Translation Completion entry, the indicated range may only be accessed using untranslated addresses and the Translated Address field of this Translation Completion entry may not be used in a subsequent Read/Write Request with AT set to *Translated*. This value may be cached if R or W is set to 1b.
R,W	**Read, Write** – These two field indicate what the transaction types that are allowed for the requests using the translation. The encodings are:

Code	Meaning
00b	Neither read nor write transactions are allowed. This translation is considered not to be valid. The contents of the Translated Address, N, and U fields are undefined. A translation with this value may not be cached in the ATC.
01b	Write Requests that target this range are allowed but Read Requests are not.
10b	Read Requests that target this range are allowed but Write Requests are not.
11b	Read and Write Requests that target this range are allowed.

2.3.1. Translation Range Size (S) Field

If S is set, then the translation applies to a range that is larger than 4KB. If S = 1 then bit 12 of the Translated Address is used to indicate whether or not the range is larger than 8KB. If bit 12 is 0, then the range size is 8KB but it is larger than 8KB if set to 1. If S = 1 and bit 12 = 1, then bit 13 is used to determine if the range is larger than 16KB or not. If bit 13 is 0 then the range size is 16KB but it is larger than 16KB if set to 1.

Low-order address bits are consumed in sequence to indicate the size of the range associated with the translation.

> Note: This encoding method is also used to indicate the size of the memory range being invalidated.

Examples for different translation sizes are shown in the following table:

Table 2-4: Examples of Translation Size Using S Field

Address Bits																						Translation Range Size in Bytes
63:32*	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	S	
x	x	X	X	x	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	4K
x	x	X	X	x	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	8K
x	x	X	X	x	x	X	x	x	x	x	x	x	x	x	x	x	x	x	0	1	1	16K
x	x	X	X	x	x	X	x	x	x	x	x	0	1	1	1	1	1	1	1	1	1	2M
x	x	X	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1G
x	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4G

*Note: Upper address bits are used to indicate the size for ranges larger than 4 GB.

The size field is set to indicate the range size in multiples of 4KB regardless of the setting of STU. For example, if STU is set to indicate that the minimum translation is 8KB, then S should be set to 1b on all translation returned in a Translation Completion and in all Invalidate Requests. If STU is set to indicate a 16KB minimum, then S and bit 12 would both be set to 1b in all translation and invalidate ranges.

2.3.2. Non-snoop (N) Field

This field is 1b to indicate that Read and Write Requests that target memory in the range of this translation must set the No Snoop Attribute bit in the Request header. When this field is 0b, the function is allowed to set the No Snoop Attribute bit in an Endpoint-specific manner. Software should insure that this bit is set only when non-snooped accesses are permissible.

When U is set to 1b, this meaning of this field is undefined and the TA may set this field to any value.

2.3.3. Untranslated Access Only (U) Field

This field is set to 1b when the Function is not allowed to access the implied range of memory using a translated address (the range is implied by the untranslated address in the Translation Request and the offset of the translation in the Translation Completion). The Function may use untranslated addresses to access the range as long as the accesses are allowed by the R and W fields. The Function may cache this translation value if either R or W is set to 1b. If the U bit is set, the Translated Address field in the translation is not necessarily a valid memory address and the Endpoint may not use the value in a Read or Write Request with AT set to *Translated*.

> Note: One of the possible uses of this field is to avoid unnecessary invalidations. If an Endpoint uses translated requests for some portions of memory but not others, then the U bit can be used on the portions for which translated requests are not used. When a translation changes, if the U bit is set, then it will not necessarily be required that an Invalidate Request be sent to the Function. An example of this use is an Endpoint with a ring buffer that is used for commands. The ring buffer may be

allocated for a long period of time and have very high re-use (locality). For this reason, it is useful for the Endpoint to use translated addresses in its memory request that target the command buffer. The same Endpoint might access data buffers that have poor locality and low reuse. Accesses to the data buffers might best be handled by using untranslated Requests. Setting the U bit for the data buffer translations insures that the Endpoint will not attempt to use a translated value to access the data buffer so, when the data buffer mappings are changed, no Invalidation Request is required.

2.3.4. Read (R) and Write (W) Fields

These fields indicate if the returned translation value may be used in a read or write memory request. The ATC may not issue a read request using the translation value if the R bit is not set to 1b. The ATC may not issue a write request using the translation value if the W bit is not set to 1b. The ATC may not issue any type of request using the translation value if neither the R nor W bits are set to 1b. If both R and W bits are set to 0b, the range of the translation is still indicated but the meaning of the other values in the translation is undefined.

Note: The range is indicated even if R=W=0b in order to allow a 'hole' in the Translation Completion. For example, if the Translation Request has a Length of 6 Dwords, then up to three translations could be included in the Translation Completion. The first and third translations may have R or W of 1b but the second could have R=W=0b. To avoid ambiguity about the size of the indicated gap, the range of the gap is indicated in the Translation Completion even if R=W=0b.

The R=0b, W=0b state is used to indicate that the address field in the translation may not be used to form a translated address value for a subsequent request.

When the host changes the translation in the TA, to make the translation present, the host is not required to send an invalidation indication to the ATC so that it will know of the change in state of the translation. Since the ATC may not be notified of changes of the translation, a translation value of R=W=0b may not be cached.

2.4. Completions with Multiple Translations

An ATC is allowed to request that the TA provide translations for a virtually contiguous range of addresses. It does this by setting the Length field in the Translation Request to a value that is two times the number of requested translations as long as the request size (Length * 4) is not larger than either Max_Read_Request_Size or RCB.

If multiple translations are requested, the TA may return one or more translations as long as the number of translations does not exceed the number of requested translations. It is not an error for the TA to return fewer translations than requested and no error indication is sent unless there is an error in accessing the data.

If the Translation Completion contains multiple translations, all translations must have the same indicated size. Also, successive translation must apply to the virtual address range that abuts the previous translation in the same completion.

If a translation has both R=0b and W=0b, the TA will must still set the Size field to the appropriate value.

Each translation in a Translation Completion will have some overlap with the implied memory range of the Translation Request (see 2.2).

A Translation Completion may require one or two CplDs.

If a Translation Completion CplD has a Byte Count that is greater than four times the Length field, then additional CplDs are required to complete the transaction.

If a Translation Completion CplD has a Byte Count that is equal to four times the Length field, then the packet completes the request. For such a CplD, if the sum of Byte Count and Lower Address is not a multiple of RCB, then the CplD is the last of a sequence and it is an error if no previous CplD has been received, and all translation values should be discarded.

> Note: There are multiple reasons that the TA may truncate the results of the completion. For example, the request might ask for a range of addresses, not all of which are defined. This could occur if the first translation were valid but was located at the end of a page of translations. The TA, in looking up the next page of translations, may find that the page is not valid so the addresses are not valid. The range of addresses that are valid would be returned and no error indicated.
>
> Note: There are multiple reasons that the TA may break a Translation Completion into multiple TLPs. As an example, if the virtual address of the Translation Completion resolves to a table access that crosses an RCB boundary of the memory system, the completion to the TA may be broken into multiple completions by the memory. Rather than require that the TA accumulate the results, it is allowed to send each portion of the Translation Completion to a Endpoint when it is received from the system memory.

3

3. Invalidation

ATS uses the messages shown in this section to maintain consistency between the TA and the ATC. This specification assumes there is a single TA associated with each ATC. The TA (in conjunction with its associated software) must ensure that the address translations cached in the ATC are not stale by issuing Invalidate Requests.

3.1. Invalidate Request

When a translation is changed in the TA and that translation might be contained within an ATC in an Endpoint, the TA (in conjunction with its associated software) must send an Invalidate Request to the ATC to maintain proper synchronization between the ATPT and the ATC. An Invalidate Request is used to clear a specific subset of the address range from the ATC. Invalidate Requests are constrained to cover power of 2 multiple of 4096 Byte pages.

The format of an Invalidation Request is:

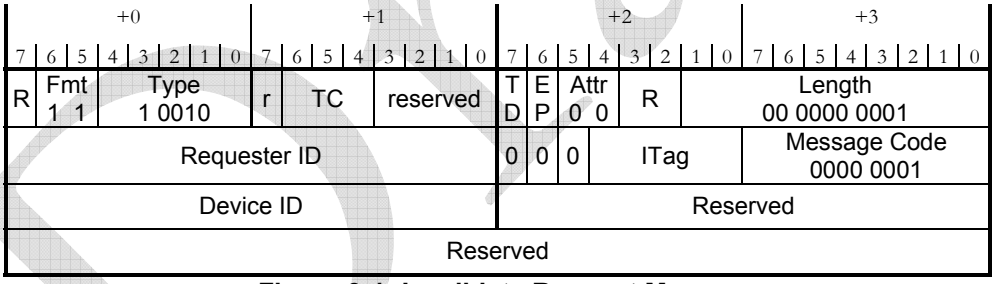


Figure 3-1: Invalidate Request Message

The Invalidate Request is a MsgD transaction with 64 bits of data. Invalidate Request messages may be sent in any TC. The ITag field is constrained to the values 0 to 31 decimal and is used by the TA to uniquely identify requests it issues. A TA must ensure that once an ITag is used, it is not reused until released by the corresponding Invalidate Completes.

The TA may have a single pool of ITag values for all invalidates that it issues or it may have a pool for each Device ID or any other combination. An Endpoint with multiple ATC on different functions must manage the ITags separately for each Requester ID

The address range specified in an Invalidate Request may span one or more STU 4096 Byte pages. Invalidation ranges are required to be naturally aligned and may not be smaller than STU 4096 Byte pages.

Given ATPT designs are implementation-specific, Invalidate Completion timeouts and the associated error handling are treated as implementation-specific and therefore are outside the scope of this specification.

Implementation Note:

Choosing a single timeout value that works in all systems may not be possible. Implementations are encouraged to provide some latitude in the completion timeout ranges supported (See section ??? of the PCI Express Base Specification).

The content of the payload is the un-translated address range to be invalidated. The payload format is:

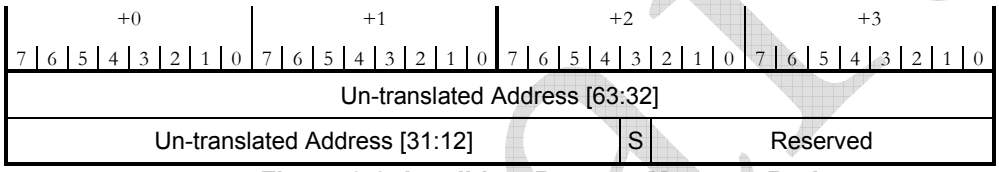


Figure 3-2; Invalidate Request Message Body

The S field is used to indicate if the range being invalidated is greater than 4096 Bytes. Its meaning is the same as for the Translation Completion (see 2.3.1). An Un-translated Address value of all 1b's indicates that the entire cache should be invalidated.

3.2. Invalidate Complete

When an Endpoint completes an Invalidate operation, it will send one or more Invalidate Complete messages to the TA. These messages will be tagged with information extracted from the Invalidate Request to enable the TA to associate the Invalidate Completes with the Invalidate Request.

The format of the Invalidate Complete Message is:

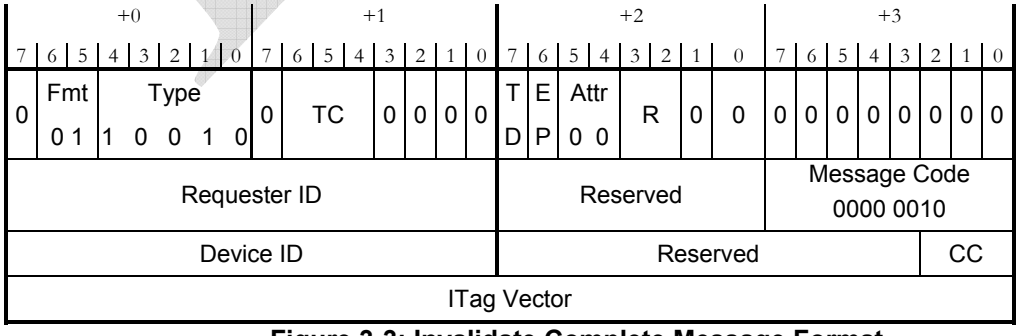


Figure 3-3: Invalidate Complete Message Format

The Invalidate Complete message is a Msg transaction routed by ID. The Requester ID field of the Invalidate Complete message is set to the Requester Id of the function containing the ATC. The Device ID field of the Invalidate Complete is set to the Requester ID of the TA. The ATC may derive the Requester ID of the TA from the Requester ID field of the corresponding Invalidate Request. Alternatively, since the ATC is only associated with a single TA , the ATC may sample and store the Requester ID from the first Invalidate Request following a Fundamental Reset or FLR. Subsequent Invalidate Complete messages may use this value to set the Device ID field of Invalidate Complete messages.

The Completion Count (CC) field indicates the number of individual Invalidate Complete messages that will be sent for the associated Invalidate Request. Setting the CC field to 0 indicates that eight responses will be sent. The TA is responsible for collecting all the responses associated with a given Tag before considering the corresponding Invalidate Request to be complete.

Invalidate Complete messages may be sent on any TC, independent of the TC the originating Invalidate Request was received. This enables implementations to utilize the Invalidate Complete to push outstanding transactions to the TA to guarantee the required invalidation semantics are met (See section ???). Implementations that utilize a single upstream TC are only required to send a single Invalidate Complete in the utilized TC.

The ITag Vector field is used to indicate which Invalidate Request has been completed. Each of the 32 possible ITag field values from the Invalidation Request is represented by a single bit in the ITag Vector field. The least significant bit, bit 0, – i.e. the right-most bit in schematic representation of Invalidate Complete Msg shown in **Error! Reference source not found.** - of the ITag Vector field corresponds to the ITag Field value of 0. The most significant bit (bit 31) of the ITag Vector field corresponds to the ITag Field value of 31. Implementations are allowed to coalesce multiple Invalidate Completions by setting multiple ITag Vector bits in a single message provided the following conditions are met:

- The Invalidate Completions flow in the same TC.
- The Invalidate Completions have the same CC value.

When combining Invalidate Completions that are replicated in multiple TC (CC not equal to 1) each message of the replicated set may be combined independent of the others in the set.

3.3. Invalidation Completion Semantics

Before an ATC can return an Invalidate Complete for a given Invalidate Request it must ensure the following conditions are satisfied:

- All new requests initiated by the Endpoint will not utilize stale address translations.
- All outstanding read requests utilizing translated address matching the invalidated range have either completed or been tagged to be discarded (method to discard is implementation specific).
- All outstanding posted writes utilizing a translated address matching the invalidated range have been pushed to the TA. The ATC is required to send a copy of the Invalidate Complete message in each TC in which a posted write has been issued but not known to have been pushed to the TA). The CC field must be set to the same value in each copy of

the Invalidate Complete message indicating number of copies sent. The TA is responsible for collecting all sent responses before considering the invalidation to be complete.

Implementation Note:

When making the decision as to which TC to send Invalidate Completions an ATC may infer, in an implementation specific manner, that an issued posted write has been pushed to the TA. For example an Endpoint that has sent a read transaction to a destination above the TA and received its corresponding response may infer that any preceding posted writes issued in the same TC have been pushed to the TA.

3.4. Request Acceptance Rules

In accord with the request acceptance rules enumerated in the PCI Express Base Specification, an Endpoint is not allowed to create a dependency in which the acceptance of a posted transaction is dependent upon the transmission of a posted transaction. Given Invalidate Requests and Invalidate Completes both are posted transactions, endpoints must not make the acceptance of an Invalidate Request dependent upon the transmission of an Invalidate Complete. The method for achieving this is implementation specific.

Implementation Note:

An ATC is only associated with a single TA. Each TA is limited to a total of 32 outstanding invalidations to any given ATC. This limits the number of outstanding Invalidation Requests active to a single ATC to 32. To avoid a post-to-post dependency, an ATC is required to implement sufficient queuing (32 entries) that such that it is capable of holding all outstanding Invalidation Requests.

An ATC may choose to implement a maximally sized input queue holding Invalidate Requests. Alternatively an ATC may choose implement a maximally sized output queue holding Invalidate Responses. Note that queuing Invalidate Responses requires significantly less state per entry resulting in a potentially more efficient implementation than input queue buffering.

Note that the choice of whether to implement input queuing or output queuing (or a hybrid of both) has no impact on ensuring deadlock free behavior. But implementation choices with regard to queuing may have a significant impact on performance (See Section 4.5).

.

A function with an ATS capability in its configuration space must be able to accept Invalidate Requests and send Invalidate Completions even if ATS is not enabled.

3.5. Invalidate Flow Control

Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A side effect of this is congestion spreading to other channels and links though the credit based flow control mechanism. Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA and its associated software implement higher level flow control mechanisms.

To assist with the implementation of Invalidate Flow Control an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the link. This field applies to all invalidations serviced by the Endpoint, independent of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability structure (see section 4.1). A value of 0 indicates that invalidate flow control is not necessary to this Endpoint.

Implementation Note:

An Endpoint may indicate that invalidate flow control is not required when one or more of the following is true:

1. The Endpoint can handle invalidations at the maximum arrival rate of Invalidate Requests.

2. The Endpoint will not or very rarely cause link backpressure (performance loss is negligible)

3. The Endpoint can fully buffer the maximum number of incoming invalidations without back pressuring the link.

3.6. Invalidate Ordering Semantics

Invalidate Requests and Translation Completions may be sent using different TC and are therefore unordered with respect to each other (from the link's perspective). An ATC must ensure that the proper invalidation behavior is maintained when an Invalidate Request bypasses a Translation Completion to an overlapping region.

An ATC must "snoop" its outstanding translation request queue against all arriving Invalidate Requests. When snooping a request for a N*STU sized translation, the ATC must snoop the range of addresses starting at the STU aligned region containing the specified address and ending (N-1) STU size pages later.

If an Invalidate Request overlaps the address range in an outstanding Translation Request, the Translation Request must be tagged as invalid and the results of its corresponding Translation Response must be discarded prior to transmission of the Invalidate Completion. If the Translation Response is received before the Invalidate Complete is sent, an implementation is free to issue requests utilizing the translation result provided the Invalidation Completion Semantics (Section 4.3) are satisfied

Implementation Note:

In the description above, N is the number of STU sized translations that were requested in the Translation Request. This is equal to (Length field in Translation Request)/2.

As an example:

STU is 00 0010b indicating 16 KB pages.

An outstanding Translation Request has a Length field of 00 0000 0100b indicating 2 translations which would cover a range of 32 KB.

The high-order 48 bits of the Translation Request are 0000 0FFF FFFFh

The low-order 16 bits of the address in the request are 11xx xxxx xxxx xxxxb indicating that the translation request covers a range that overlaps a 32KB boundary (in fact, the request crosses a 16TB boundary) .

If two translations are returned, they would cover the two STU sized regions at 0000 0FFF FFFF C000h and 0000 1000 0000 0000h

An Invalidate Request is received with the high-order 48 bits of 0000 1000 0000h and the low-order 16 bits of 0001 1xxx xxxx xxxxb.

The ATC must detect that a translation associated with a portion of the Translation Request is now invalidated and the Translation Completion associated with the invalidated region must be discarded (for simplification, the ATC is allowed to discard all of the Translation Completion).

It should be noted that processing of the Invalidates is simplified if Translation Requests do not cross alignment boundaries of the request. The Translation Request from the above example is not aligned to a 32 KB boundary. If it were broken into two requests, it would be simpler to associate the range of the Invalidate Request with the address in the Translation Request. Breaking the Translation Requests into aligned requests is not a requirement.

3.7. Implicit Invalidation Events

The following events will cause the invalidation of all ATC entries:

- Fundamental Reset (all forms)
- Function Level Reset

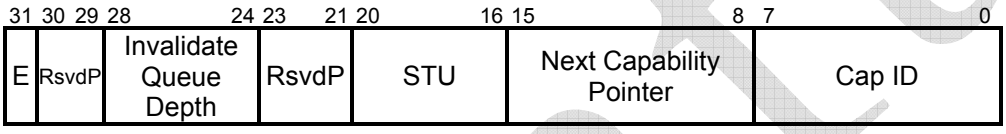
No explicit Invalidate Complete is sent when the invalidation is implicit.

4

4. Configuration

4.1. ATS Capability Structure

Each Function that supports ATS must have the ATS Capability structure in its configuration space. The format for the capability is:



Bit Location	Register Description	Attributes
7:0	**Capability ID** – Indicates the ATS Capability structure. This field must return a Capability ID of xxh **[TBD]** indicating that this is an ATS Capability structure.	RO
15:8	**Next Capability Pointer** – The offset to the next PCI capability structure or 00h if no other items exist in the linked list of capabilities.	RO
20:16	**Smallest Translation Unit (STU)** – This value indicates to the Function the minimum number of 4KB blocks that will indicated in a Translation Completion or Invalidate Requests. This is a power of 2 multiplier and the number of blocks is 2^STU. A value of 0 indicates one 4KB block and a value of 11111b would indicate an 8TB block. Default value is 00000b.	RW
28:24	**Invalidate Queue Depth** – The number of Invalidate Requests that the Endpoint can accept before putting backpressure on the upstream connection. If 0, the Endpoint can accept 32 Invalidate Requests.	RO
31	**Enable (E)** – When set, the Endpoint is enabled to cache translations. Default value is 0	RW

Figure 4-1 ATS Capability Structure

5

5. Acknowledgements

The following people were instrumental in the development of the ATS Specification[1].

[Send additional names to opt in for this section to Mike]

Andrew Gruber, ATI Corporation

Mark Hummel, Advanced Micro Devices, Inc.

Michael Krause, Hewlett-Packard

Brian Langendorf, Nvidia Corporation

Rajesh Sankaran, Intel Corporation

David Wooten, Microsoft Corporation

[1] Company affiliation is at the time of specification contribution.